SISTEMAS BASADOS EN MICROPROCESADOR **Septiembre 2021/2022**

Apellidos:.....SOLUCIÓN.....

Nombre:..... Puesto:.....

P1	P2	P3

Duración 4:00 horas

1.- [3 puntos] Preguntas cortas:

a) Considere la instrucción `add.w &P1IV, PC`. Ensámblela y diga cuántos ciclos tarda en ejecutarse (NOTA: considere `P1IV`=0x210).

b) Suponga que se produce una IRQ cuando la CPU está en un modo de bajo consumo. Indique qué sucede antes de la ejecución de la ISR y después del RETI. ¿Qué debería hacer en la ISR para salir del modo de bajo consumo y volver al programa principal?

c) Al arrancar la frecuencia del ACLK es de 37'5 kHz aproximadamente. Justifíquelo.

d) Escriba las instrucciones necesarias para poner la CPU a 7MHz.

SOLUCIÓN

a)

```
;                                Emulación            Código máquina
;                                                     OpCd -Rf- DdBDf -Rd-(tipo I)
;                                                     000100 OpC BDf -Rd-(tipo II)
;                                                     001 Cnd Desplazami(salto)
;-------------------------------------------------------------------------------
           add.w  &0x201, PC    ;                     0101 0010 0001 0000=5210 0210
```

La instrucción ocupa 2 palabras y tarda 4 ciclos en ejecutarse, 3 ciclos según los modos de direccionamiento, más 1 por ser el PC el registro destino. Los ciclos serían:

- Leer la instrucción.
- Leer la dirección del operando fuente.
- Leer el operando fuente.
- Guardar el resultado en el PC.

b) Las acciones que se toman ANTES son:

- Se poner la CPU en modo activo.
- Se apilan el PC y el SR.
- Se borra el SR.
- Se selecciona la IRQ más prioritaria. Si es una interrupción con una fuente única, se borra el flag.
- Se carga en el PC el vector de intrerrupción asociado a la IRQ leyéndolo de la tabla de vectores de interrupción, provocando que se salte al inicio de la ISR.

Las acciones que se toman DESPUÉS son:

- Al hacer RETI, se desapilan el PC y el SR, con lo que la CPU vuelve al modo en el que estuviera antes de la IRQ. En nuestro caso, como partíamos de un modo de bajo consumo, se vuelve al mismo.

Cómo salir del modo de bajo consumo y volver al programa principal:

- Borrar los bits de bajo consumo del SR que se encuentra en la pila:

```
bic.w  #LMP4, 0(SP)  ;Volver al modo activo después de la ISR
```

c) El valor por defecto del campo `SELA` (registro `CSCTL2`) es 000 (LFXTCLK). El campo `LFXTOFF` en `CSCTL4` es 1 (LFXT apagado salvo si se usa como fuente de algún reloj). Como se usa para el ACLK, no debería estar apagado. El problema es que para que el LFXT arranque es necesario que sus pines se conecten con el cristal de cuarzo que hay en el exterior. Por defecto, TODOS los pines del FR6989 arrancan en modo de alta impedancia por lo que el driver del reloj no se encuentra conectado, y por tanto, el reloj no puede arrancar. Cuando se produce un fallo en el LFXT, el módulo de reloj conecta automáticamente el reloj a la salida de baja frecuencia del MODCLK (que tiene una frecuencia nominal de 4'8MHz). Dicha salida es el propio MODCLK dividido por 128. De ahí

que la frecuencia del ACLK al arrancar sea 4'8MHz/128=37'5kHz.

d)

```
mov.b  #CSKEY_H, &CSCTL0_H;Desbloquear módulo de reloj
xor.w  #DCOFSEL1|DCOFSEL0, &CSCTL1;Pasar de modo 6(8MHz) a 5 (7MHz)
bic.w  #DIVM2|DIVM1|DIVM0, &CSCTL3;DIVM=0 (dividir por 1)
clr.b  &CSCTL0_H     ;Bloquear módulo de reloj
```

CRITERIO DE CORRECCIÓN

a) Ensamblar instrucción: 20%
b) Antes y después de ISR en bajo consumo: 30% (antes 5, después 3, salir 2)
c) Por qué f_{ACLK}=37'5kHz 20%
d) Hacer f_{MCLK}=7MHz 30% (4 cambiar DCOFSEL, 4 DIVM y 2 bloqueo)

2.- [3 puntos] Realizar la función *strtrim* en ensamblador del MSP430 que seguirá la convención de llamada tipo C y que atenderá al siguiente prototipo:

```
void strtrim(char *s);
```

donde *s* es un puntero a una cadena de caracteres tipo C. La subrutina *strtrim* debe eliminar los espacios iniciales y finales de la cadena. Por ejemplo, si la función es llamada con la cadena "·····*Hola*···*mundo*·····" (donde el carácter '·' simboliza el carácter espacio), la dejaría como "*Hola*···*mundo*".

a) Explique con palabras el algoritmo que usará para resolver el problema.
b) Escriba la subrutina en ensamblador.

SOLUCIÓN

a) Para eliminar los espacios iniciales y finales de una cadena se puede proceder como sigue:

- Buscar el final de la cadena (carácter NULO) y situar un puntero (*p*) en el final.
- Retroceder p en dirección al principio mientras el carácter sea blanco.
- Si se ha encontrado un carácter no blanco, escribir después un NULO.
- Desde el principio de la cadena, buscar el primer carácter no blanco. Supongamos que su dirección está en *p*.
- Realizar una copia de la cadena apuntada por *p* a *s*.

b) Casos de especial interés para comprobar:

- Cadena vacía "".
- Cadena blanca "····".
- Cadena con espacios finales "Hola···".
- Cadena con espacios iniciales "·····Hola".
- Cadena con espacios centrales "Hola·····mundo".

```
;-------------------------------------------------------------------------------
; void strtrim (char *s);                                                 v1.0
;
; Elimina de s los espacios iniciales y finales.
;-------------------------------------------------------------------------------
strtrim    ; Buscar el final de la cadena
           mov.w  r12, r13      ;r13=s. Apuntará al final de s
strtrim1   mov.b  @r13+, r14    ;Leer carácter y avanzar puntero
           tst.b  r14           ;Es fin de cadena?
           jne    strtrim1      ;...no. Seguir buscando
           dec.w  r13           ;r13 apunta al final de la cadena (sobre el nulo)
           cmp.w  r12, r13      ;Si r12=r13, era cadena vacía
           jeq    strtrimfin

           ; Buscar espacios desde el final hasta llegar al principio de la cadena
           mov.b  @r12, r15     ;Guardar primer carácter...
           clr.b  0(r12)        ;...porque se usará como centinela
strtrim2   dec.w  r13           ;Retroceder puntero
           mov.b  @r13, r14     ;Leer carácter del final
           cmp.b  #' ', r14     ;Es blanco?
           jeq    strtrim2      ;...sí, seguir
           clr.b  1(r13)        ;Poner final de cadena (eliminar espacios finales)
           mov.b  r15, 0(r12)   ;Restaurar primer carácter
```

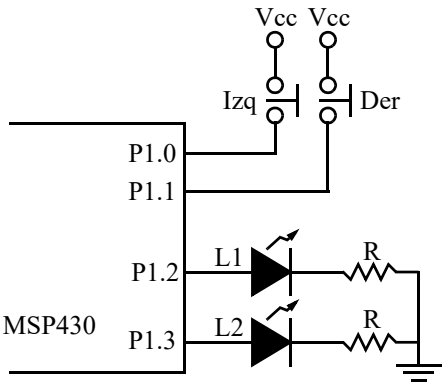
```
            ;Buscar primer carácter no blanco desde el principio
            mov.w  r12,r13       ;r13 apuntará al primer carácter no blanco
strtrim3    mov.b  @r13+, r14    ;Leer carácter y avanzar puntero
            cmp.b  #' ', r14     ;Es blanco?
            jeq    strtrim3      ;...sí, seguir buscando
            dec.w  r13           ;...no.r13 apunta primer no blanco (puede que NUL)
            cmp.w  r12, r13      ;r12=r13?
            jeq    strtrimfin    ;...sí. No había espacios iniciales. Terminado

            ;Eliminar blancos iniciales (copiar subcadena)
strtrim4    mov.b  @r13+, r14    ;Leer carácter a mover y avanzar puntero fuente
            mov.b  r14, 0(r12)   ;Escribir en destino
            inc.w  r12           ;Avanzar puntero destino
            tst.b  r14           ;Era el final de cadena?
            jne    strtrim4      ;...no. Seguir. Se sale cuando se compia el '\0'

strtrimfin ret
```

3.- [4 puntos] Considere el circuito de la figura que corresponde con el circuito eléctrico del mecanismo de luces de dirección (intermitentes) de un vehículo. La palanca de los intermitentes actúa sobre `Izq` cuando se acciona hacia la izquierda y sobre `Der` cuando se hace a la derecha. La palanca garantiza que no es posible activar ambos pulsadores a la vez. Un mecanismo de enclavamiento, percibido por el conductor como una pequeña resistencia al desplazamiento que debe vencer, bloquea la palanca en la posición deseada hasta que el conductor la libera manualmente o se gira el volante en dirección contraria. Cuando se actúa sobre la palanca sin forzar la resistencia (modo confort), la misma retorna a la posición de reposo al liberarla. Haga un programa en ensamblador del MSP430 que gestione en el mejor modo de bajo consumo, con el TA0 y ***por interrupciones*** el encendido/apagado de los leds en función de los pulsadores:

- Cuando se activa uno de los pulsadores, se enciende el led correspondiente con una frecuencia de 1 Hz y un ciclo de trabajo del 50% (0'5s encendido, 0'5s apagado) hasta que es liberado. No obstante, siempre se hacen al menos 3 ciclos de apagado/ encendido a pesar que de la pulsación sea inferior a 3s (mecanismo de confort).
- Si se acciona el pulsador contrario, se corta inmediatamente la intermitencia actual y se inicia en la dirección contraria.

Notas:

- Considere los pulsadores libres de rebotes, el perro guardián desactivado, los puertos desbloqueados, pila inicializada y ACLK=LFXT en marcha con cristal de 32768Hz.
- Los puertos P1.0 a P1.3 están conectados a los CCR0 a CCR3 del TA0 en su función primaria. Entradas por CCIA.



SOLUCIÓN

Dado que los pulsadores no tienen resistencias externas, hay que usar las internas. En este caso deben ser de *pull-down*. La tecla pulsada se detectará con un valor alto (lógica positiva). El control de los leds es con lógica positiva (1 enciende, 0 apaga).

Todos los puertos se pondrán en su función primaria para que sean controlados por el TA0 directamente. Los CCR0 y CCR1 se configurarán en modo de captura sensible a flanco de

subida en el canal CCIA con las interrupciones habilitadas. Los CCR2 y CCR3 se configurarán en modo de comparación con la salida a 0 (intermitentes apagados) sin IRQ.

El TA0 se configurará con entrada de ACLK, en modo continuo con el divisor máximo (64), ya que sólo hay que medir tiempos de 0'5s. Eso nos da que el TA0 tardará $2^{16}/(2^{15}/64)$=128s en desbordarse. Este tiempo es el máximo que se podrá medir en el modo captura, que es el que está configurado en los canales 0 y 1. Si la palanca de los intermitentes estuviera accionada más de ese tiempo (2 minutos y 8 segundos), podríamos tener problemas.

Los procesos de los canales izquierdo y derecho son análogos. Para evitar repetir el código en ambos canales, el mismo se ha parametrizado y se han usado los registros R5 a R10 para guardar las direcciones de los registros del TA0 que se tienen que usar. En esta descripción se va a suponer que se ha pulsado la palanca a la izquierda, pero sería análogo para la derecha.

El CCR0 controla la palanca izquierda. Puede estar en tres modos distintos:

- Captura, sensible a flanco de subida (modo inicial). Se espera a la pulsación de la tecla. En la ISR se cancela la posible intermitencia del lado derecho y se inicia la del lado izquierdo:
  - Se apaga el intermitente derecho por si estaba encendido (CCR3, `OUTMOD_0` y `OUT=0`).
  - Se interrumpe un posible evento de comparación para tiempo de confort en el lado derecho (CCR1 en modo captura sensible a flanco de subida).
  - Se enciende el intermitente izquierdo (CCR2, `OUTMOD_0` y `OUT=1`).
  - Se lee el instante de pulsación de la tecla.
  - Se programa el CCR2 para un evento de comparación con IRQ dentro de 0'5 segundos (el semiperiodo de la intermitencia) en el que se conmutará la salida.
  - Se calcula el tiempo de fin de confort como 5 semiperiodos desde la pulsación (deberían ser 6, pero como el último semiperiodo es apagado, se simplifica). Este tiempo se guarda en la variable `TFinConf`.
  - Se cambia a flanco de bajada para esperar la liberación de la tecla.
- Captura, sensible a flanco de bajada. Se espera a la liberación de la tecla. En la ISR se mira si el tiempo actual (capturado en el CCR0) es mayor o igual que el instante final de confort (guardado en la variable `TFinConf`). Si es así, se apaga la intermitencia (ver punto siguiente). Si no, hay que esperar el tiempo de confort. Para ello se pasa el CCR0 a modo de comparación para que produzca un evento (sólo la IRQ) al final de dicho tiempo.
- Comparación. Se espera al fin del tiempo de confort. En la ISR se apaga el intermitente izquierdo y se configura el CCR0 en modo captura sensible en flanco de subida. Durante este modo, y mientras no se cumpla el tiempo, la tecla Izq no está operativa. Por ello, sucesivas maniobras sobre ella son ignoradas.

El CCR1 es análogo al CCR0, pero para la tecla derecha.

En las ISR de CCR2 y CCR3 simplemente se actualizan los tiempos del semiperiodo de intermitencia de cada lado. La conmutación es automática.

```
IZQ         .equ   BIT0
DER         .equ   BIT1
L1          .equ   BIT2
L2          .equ   BIT3
TECLAS      .equ   IZQ|DER
LEDS        .equ   L1|L2

FACLK       .equ   32768          ;Frecuencia de ACLK. En Hz
DIVTA       .equ   64             ;Divisor del TA0
FTA         .equ   FACLK/DIVTA    ;Frecuencia del TA0. En Hz
FINTER      .equ   1              ;Frecuencia de los intermitentes. En Hz
CCRINTER    .equ   FTA/(2*FINTER)-1;Valor inicial de CCR2 y CCR3
TCONFORT    .equ   5*FINTER       ;Tiempo mínimo intermitente activos (en TICS)

            .bss   TFinConf, 2    ;Variable que guarda el tiempo fin de confort

;-------------------------------------------------------------------------------
```

```
;main                                                                                    v1.0
;-----------------------------------------------------------------------------------------
main        ;P1.2 y P1.3 salida función 1 (TA0.2 y TA0.3)
            bis.b  #LEDS, &P1DIR

            ;P1.0 y P1.1 entrada pulldown función 1 (TA0.0 y TA0.1)
            bis.b  #TECLAS, &P1REN;Resistencia...
            bic.b  #TECLAS, &P1OUT;...de pulldown
            bis.b  #LEDS|TECLAS, &P1SEL0;Función 1 para ambos

            ;TA0.2 y TA0.3 en modo comparación con salida a 0 (sin intermitencia)
            mov.w  #OUTMOD_0, &TA0CCTL2
            mov.w  #OUTMOD_0, &TA0CCTL3

            ;TA0.0 y TA0.1 en modo captura con entrada CCIA en flanco de subida. IRQ
            mov.w  #CAP|CM_1|CCIS_0|CCIE, &TA0CCTL0
            mov.w  #CAP|CM_1|CCIS_0|CCIE, &TA0CCTL1

            ;TA0 con ACLK/64 modo continuo
            mov.w  #7, &TA0EX0   ;Divisor extra a 8
            mov.w  #TASSEL__ACLK|ID__8|MC__CONTINUOUS|TACLR, &TA0CTL
            bis.w  #LPM3|GIE, sr ;Ea, a dormir

;-----------------------------------------------------------------------------------------
; TA01ISR                                                                          v1.0
;-----------------------------------------------------------------------------------------
TA01ISR     add.w  &TA0IV, PC    ;Saltar a ISR. Borra flag
            reti                 ;IV=0 No IRQ
            jmp    TA01CCR1      ;IV=2. CCR1
            jmp    TA01CCR2      ;IV=4. CCR2
            ;jmp   TA01CCR3      ;IV=6. CCR3

;**** CCR3 ****
            ;Rutina de servicio de la IRQ del CCR3 (intermitente derecho)
TA01CCR3    add.w  #CCRINTER, &TA0CCR3
            reti

;**** CCR2 ****
            ;Rutina de servicio de la IRQ del CCR2 (intermitente izquierdo)
TA01CCR2    add.w  #CCRINTER, &TA0CCR2
            reti

;**** CCR1 ****
TA01CCR1    ;Rutina de servicio de la IRQ del CCR1 (palanca derecha)
            push   r4

            ;Tres posibles eventos de entrada:
            ;1. Flanco de subida en CCIA. Se ha pulsado la tecla
            ;2. Flanco de bajada en CCIA. Se ha soltado la tecla
            ;3. Evento de comparación. Se ha terminado el tiempo de confort
            bit.w  #CAP, &TA0CCTL1;Qué modo está programado?
            jz     Comparac      ;...comparación, apagar intermitente
            bit.w  #CM0, &TA0CCTL1;...captura. Qué flanco está programado?
            jz     FBajada       ;...bajada. Se ha soltado la tecla
            ;Si flanco subida, tomar tiempo de inicio y activar intermitente
FSubida1    ;Cancelar posible parpadeo en el lado contrario
            mov.w  #OUTMOD_0, &TA0CCTL2;Apagar posible parp. en lado contrario. No IRQ
            mov.w  #CAP|CM_1|CCIS_0|CCIE, &TA0CCTL0;Desact. pos. TFinConf lado opuesto
            ;Iniciar parpadeo en este lado
            mov.w  #OUTMOD_0|OUT, &TA0CCTL3;Encender intermitente de este lado
            mov.w  &TA0CCR1, r4  ;R4=Instante de pulsación
            add.w  #CCRINTER, r4 ;R4=Instante de conmutación
            mov.w  r4, &TA0CCR3  ;Programar tiempo de siguiente semiciclo
            mov.w  #OUTMOD_4|CCIE, &TA0CCTL3;Conmutar en siguiente semiciclo. IRQ
            add.w  #TCONFORT-CCRINTER, r4;R4=Instante final del tiempo de confort
            mov.w  r4, &TFinConf ;Recordar ese tiempo
            xor.w  #CM1|CM0, &TA0CCTL1;Conmutar flanco de entrada (sensible a bajada)
            jmp    TA01Fin
```

```
            ;Si es flanco de bajada, ver si se ha cumplido el tiempo de confort
FBajada1    cmp.w  &TA0CCR1, &TFinConf;Hemos llegado al tiempo mínimo?
            jlo    Comparac     ;...sí. Apagar intermitente
            ;...no. Programar tiempo de fin
            mov.w  &TFinConf, &TA0CCR1;Programar tiempo de apagado
            mov.w  #CCIE, &TA0CCTL1;Pedir una IRQ en ese instante
            jmp    TA01Fin
            ;Evento de comparación, apagar intermitente y ser sensible a tecla
Comparac1   mov.w  #OUTMOD_0, &TA0CCTL3;Apagar intermitente, sin IRQ
            mov.w  #CAP|CM_1|CCIS_0|CCIE, &TA0CCTL1;Sensible a tecla
TA01Fin     pop    r4
            reti

;-------------------------------------------------------------------------------
; TA00ISR                                                                 v1.0
;-------------------------------------------------------------------------------
TA00ISR     ;Rutina de servicio de la IRQ del CCR1 (palanca izquierda)
            push   r4

            ;Tres posibles eventos de entrada:
            ;1. Flanco de subida en CCIA. Se ha pulsado la tecla
            ;2. Flanco de bajada en CCIA. Se ha soltado la tecla
            ;3. Evento de comparación. Se ha terminado el tiempo de confort
            bit.w  #CAP, &TA0CCTL0;Qué modo está programado?
            jz     Comparac     ;...comparación, apagar intermitente
            bit.w  #CM0, &TA0CCTL0;...captura. Qué flanco está programado?
            jz     FBajada      ;...bajada. Se ha soltado la tecla
            ;Si flanco subida, tomar tiempo de inicio y activar intermitente
FSubida0    ;Cancelar posible parpadeo en el lado contrario
            mov.w  #OUTMOD_0, &TA0CCTL3;Apagar posible parp. en lado contrario. No IRQ
            mov.w  #CAP|CM_1|CCIS_0|CCIE, &TA0CCTL1;Desact. pos. TFinConf lado opuesto
            ;Iniciar parpadeo en este lado
            mov.w  #OUTMOD_0|OUT, &TA0CCTL2;Encender intermitente de este lado
            mov.w  &TA0CCR0, r4  ;R4=Instante de pulsación
            add.w  #CCRINTER, r4 ;R4=Instante de conmutación
            mov.w  r4, &TA0CCR2  ;Programar tiempo de siguiente semiciclo
            mov.w  #OUTMOD_4|CCIE, &TA0CCTL2;Conmutar en siguiente semiciclo. IRQ
            add.w  #TCONFORT-CCRINTER, r4;R4=Instante final del tiempo de confort
            mov.w  r4, &TFinConf;Recordar ese tiempo
            xor.w  #CM1|CM0, &TA0CCTL0;Conmutar flanco de entrada (sensible a bajada)
            jmp    TA00Fin
            ;Si es flanco de bajada, ver si se ha cumplido el tiempo de confort
FBajada0    cmp.w  &TA0CCR0, &TFinConf;Hemos llegado al tiempo mínimo?
            jlo    Comparac     ;...sí. Apagar intermitente
            ;...no. Programar tiempo de fin
            mov.w  &TFinConf, &TA0CCR0;Programar tiempo de apagado
            mov.w  #CCIE, &TA0CCTL0;Pedir una IRQ en ese instante
            jmp    TA00Fin
            ;Evento de comparación, apagar intermitente y ser sensible a tecla
Comparac0   mov.w  #OUTMOD_0, &TA0CCTL2;Apagar intermitente, sin IRQ
            mov.w  #CAP|CM_1|CCIS_0|CCIE, &TA0CCTL0;Sensible a tecla
TA00Fin     pop    r4
            reti

            .intvecTIMER0_A0_VECTOR, TA00ISR
            .intvecTIMER0_A1_VECTOR, TA01ISR
            .intvecRESET_VECTOR, main
```

Las subrutinas de CCR0 y CCR1 son esencialmente idénticas, salvo por los registros sobre los que se actúa. Esto supone una dificultad de mantenimiento, ya que habría que asegurarse de modificar ambas secciones del código. A continuación se muestra una versión parametrizada del código. En lugar de acceder directamente a los registros implicados, las direcciones de los mismos se meten primero en registros de la CPU y hay un único código para CCR0 y CCR1 que realizar la tarea.

```
;-------------------------------------------------------------------------------
; TA01ISR                                                                 v1.1
;-------------------------------------------------------------------------------
```

```
TA01ISR     add.w  &TA0IV, PC    ;Saltar a ISR. Borra flag
            reti                ;IV=0 No IRQ
            jmp    TA01CCR1     ;IV=2. CCR1
            jmp    TA01CCR2     ;IV=4. CCR2
            ;jmp   TA01CCR3     ;IV=6. CCR3

;**** CCR3 ****
            ;Rutina de servicio de la IRQ del CCR3 (intermitente derecho)
TA01CCR3    add.w  #CCRINTER, &TA0CCR3
            reti

;**** CCR2 ****
            ;Rutina de servicio de la IRQ del CCR2 (intermitente izquierdo)
TA01CCR2    add.w  #CCRINTER, &TA0CCR2
            reti

;**** CCR1 ****
TA01CCR1    ;Rutina de servicio de la IRQ del CCR1 (palanca derecha)
            push   r4
            push   r5
            push   r6
            push   r7
            push   r8
            push   r9
            push   r10
            mov.w  #TA0CCTL1, r5;R5=Registro de control de tecla derecha
            mov.w  #TA0CCR1, r6 ;R6=CCR de tecla derecha
            mov.w  #TA0CCTL3, r7;R7=Registro de control de intermitente derecho
            mov.w  #TA0CCR3, r8 ;R8=CCR de intermitente derecho
            mov.w  #TA0CCTL2, r9;R9=Registro de control del intermitente izqdo
            mov.w  #TA0CCTL0, r10;R10=Registro de control de tecla izqda

            ;Código oomún para teclas izquierda y derecha
            ;Tres posibles eventos de entrada:
            ;1. Flanco de subida en CCIA. Se ha pulsado la tecla
            ;2. Flanco de bajada en CCIA. Se ha soltado la tecla
            ;3. Evento de comparación. Se ha terminado el tiempo de confort
CCR0y1      bit.w  #CAP, 0(r5)  ;Qué modo está programado?
            jz     Comparac     ;...comparación, apagar intermitente
            bit.w  #CM0, 0(r5)  ;...captura. Qué flanco está programado?
            jz     FBajada      ;...bajada. Se ha soltado la tecla
            ;Si flanco subida, tomar tiempo de inicio y activar intermitente
FSubida     ;Cancelar posible parpadeo en el lado contrario
            mov.w  #OUTMOD_0, 0(r9);Apagar posible parpadeo en lado contrario. No IRQ
            mov.w  #CAP|CM_1|CCIS_0|CCIE, 0(r10);Desact. posible TFinConf lado opuesto
            ;Iniciar parpadeo en este lado
            mov.w  #OUTMOD_0|OUT, 0(r7);Encender intermitente de este lado
            mov.w  @r6, r4      ;R4=Instante de pulsación
            add.w  #CCRINTER, r4;R4=Instante de conmutación
            mov.w  r4, 0(r8)    ;Programar tiempo de siguiente semiciclo
            mov.w  #OUTMOD_4|CCIE, 0(r7);Conmutar en siguiente semiciclo. IRQ
            add.w  #TCONFORT-CCRINTER, r4;R4=Instante final del tiempo de confort
            mov.w  r4, &TFinConf;Recordar ese tiempo
            xor.w  #CM1|CM0, 0(r5);Conmutar flanco de entrada (sensible a bajada)
            jmp    TA01Fin
            ;Si es flanco de bajada, ver si se ha cumplido el tiempo de confort
FBajada     cmp.w  @r6, &TFinConf;Hemos llegado al tiempo mínimo?
            jlo    Comparac     ;...sí. Apagar intermitente
            ;...no. Programar tiempo de fin
            mov.w  &TFinConf, 0(r6);Programar tiempo de apagado
            mov.w  #CCIE, 0(r5) ;Pedir una IRQ en ese instante
            jmp    TA01Fin
            ;Evento de comparación, apagar intermitente y ser sensible a tecla
Comparac    mov.w  #OUTMOD_0, 0(r7);Apagar intermitente, sin IRQ
            mov.w  #CAP|CM_1|CCIS_0|CCIE, 0(r5);Sensible a tecla
TA01Fin     pop    r10
            pop    r9
            pop    r8
```

```
            pop    r7
            pop    r6
            pop    r5
            pop    r4
            reti

;-------------------------------------------------------------------------------
; TA00ISR                                                                  v1.1
;-------------------------------------------------------------------------------
TA00ISR     ;Rutina de servicio de la IRQ del CCR0 (palanca izquierda)
            push   r4
            push   r5
            push   r6
            push   r7
            push   r8
            push   r9
            push   r10
            mov.w  #TA0CCTL0, r5 ;R5=Registro de control de tecla izquierda
            mov.w  #TA0CCR0, r6  ;R6=CCR de tecla izquierda
            mov.w  #TA0CCTL2, r7 ;R7=Registro de control de intermitente izquierdo
            mov.w  #TA0CCR2, r8  ;R8=CCR de intermitente izquierdo
            mov.w  #TA0CCTL3, r9 ;R9=Registro de control del intermitente derecho
            mov.w  #TA0CCTL0, r10;R10=Registro de control de tecla derecha
            jmp    CCR0y1        ;Ir a proceso común de ambos intermitentes
```

El código resultante evita repetir la sección crítica (unas 52 palabras), pero requiere unas 42 palabras extra para salvar/recuperar los registros de la CPU y la carga de las direcciones de los registros del TA0. Además, tarda 24 ciclos extra en hacer ese trabajo. También es más difícil de seguir, ya que hay que estar siguiendo la pista de a dónde apunta cada registro.

Posiblemente la mejor opción sea usar las capacidades del ensamblador para que sea él quien parametrice el código. El objeto en cuestión se llama macro y es un potente mecanismo de sustitución de código que permite incluir parámetros y otras herramientas. Primero se define la macro incluyendo el código de sustitución entre las directivas `.macro` y `.endm`. La etiqueta que precede a `.macro` es el nombre de la misma y le siguen los posibles parámetros formales. Después se "llama" la macro en los puntos deseados poniendo la etiqueta del nombre (`Intermit`) seguida de los parámetros reales. El ensamblador "expande" la macro sustituyendo los parámetros formales por los reales. Consulte el manual del ensamblador para más detalles.
Este mecanismo es distinto a una llamada a subrutina. En ésta sólo hay una copia del código. En cambio, como la macro es usada dos veces, el código aparece dos veces, adaptado a cada canal, tal como se hizo manualmente en la primera versión de la solución. Pero al igual que con la subrutina, sólo hay un código que mantener, que se puede adaptar al canal que se quiera y sin la sobrecarga que supone una subrutina (no hay que usar registros, no hay que guardarlos en la pila,...). Se usa un parámetro para personalizar las etiquetas de salto, ya que no se pueden repetir las mismas.

```
;-------------------------------------------------------------------------------
; Macro Intermit
; Crea el código común para los CCR0 y CCR1 del problema de los internitentes
; Los parámetros son:
; n: Un número que se anexa a las etiquetas para hacerlas disintas
; CCTLel:  CCTL del canal de entrada del lado en cuestión
; CCRel:   CCR del canal de entrada del lado en cuestión
; CCTLsl:  CCTL del canal de salida del lado en cuestión
; CCRsl:   CCR del canal de salida del lado en cuestión
; CCTLeo:  CCTL del canal de entrada del otro lado
; CCReo:   CCR del canal de entrada del otro lado
; CCTLso:  CCTL del canal de salida del otro lado
; CCRso:   CCR del canal de salida del otro lado
;-------------------------------------------------------------------------------
Intermit    .macro n,CCTLel,CCRel,CCTLsl,CCRsl,CCTLeo,CCReo,CCTLso,CCRso
            push   r4

            ;Tres posibles eventos de entrada:
```

```
            ;1. Flanco de subida en CCIA. Se ha pulsado la tecla
            ;2. Flanco de bajada en CCIA. Se ha soltado la tecla
            ;3. Evento de comparación. Se ha terminado el tiempo de confort
            bit.w  #CAP, &CCTLel ;Qué modo está programado?
            jz     Comparac:n:   ;...comparación, apagar intermitente
            bit.w  #CM0, &CCTLel ;...captura. Qué flanco está programado?
            jz     FBajada:n:    ;...bajada. Se ha soltado la tecla
            ;Si flanco subida, tomar tiempo de inicio y activar intermitente
FSubida:n:  ;Cancelar posible parpadeo en el lado contrario
            mov.w  #OUTMOD_0, &CCTLso;Apagar posible parp. en lado contrario. No IRQ
            mov.w  #CAP|CM_1|CCIS_0|CCIE, &CCTLeo;Desact. pos. TFinConf lado opuesto
            ;Iniciar parpadeo en este lado
            mov.w  #OUTMOD_0|OUT, &CCTLsl;Encender intermitente de este lado
            mov.w  &CCRel, r4    ;R4=Instante de pulsación
            add.w  #CCRINTER, r4 ;R4=Instante de conmutación
            mov.w  r4, &TCCRsl   ;Programar tiempo de siguiente semiciclo
            mov.w  #OUTMOD_4|CCIE, &CCTLsl;Conmutar en siguiente semiciclo. IRQ
            add.w  #TCONFORT-CCRINTER, r4;R4=Instante final del tiempo de confort
            mov.w  r4, &TFinConf ;Recordar ese tiempo
            xor.w  #CM1|CM0, &CCTLel;Conmutar flanco de entrada (sensible a bajada)
            jmp    Fin:n:
            ;Si es flanco de bajada, ver si se ha cumplido el tiempo de confort
FBajada:n:  cmp.w  &CCRsl, &TFinConf;Hemos llegado al tiempo mínimo?
            jlo    Comparac:n:   ;...sí. Apagar intermitente
            ;...no. Programar tiempo de fin
            mov.w  &TFinConf, &CCRsl;Programar tiempo de apagado
            mov.w  #CCIE, &CCTLel;Pedir una IRQ en ese instante
            jmp    Fin:n:
            ;Evento de comparación, apagar intermitente y ser sensible a tecla
Comparac:n:mov.w  #OUTMOD_0, &CCTLsl;Apagar intermitente, sin IRQ
            mov.w  #CAP|CM_1|CCIS_0|CCIE, &CCTLel;Sensible a tecla
Fin:n:      pop    r4
            reti
            .endm
;-------------------------------------------------------------------------------
; TA01ISR                                                                 v1.2
;-------------------------------------------------------------------------------
TA01ISR     add.w  &TA0IV, PC    ;Saltar a ISR. Borra flag
            reti                 ;IV=0 No IRQ
            jmp    TA01CCR1      ;IV=2. CCR1
            jmp    TA01CCR2      ;IV=4. CCR2
            ;jmp   TA01CCR3      ;IV=6. CCR3

;**** CCR3 ****
            ;Rutina de servicio de la IRQ del CCR3 (intermitente derecho)
TA01CCR3    add.w  #CCRINTER, &TA0CCR3
            reti

;**** CCR2 ****
            ;Rutina de servicio de la IRQ del CCR2 (intermitente izquierdo)
TA01CCR2    add.w  #CCRINTER, &TA0CCR2
            reti

;**** CCR1 ****
TA01CCR1    ;Rutina de servicio de la IRQ del CCR1 (palanca derecha)
            Intermit    1,TA0CCTL1,TA0CCR1,TA0CCTL3,TA0CCR3,TA0CCTL0,TA0CCR0,TA0CCTL2,
TA0CCR2

;-------------------------------------------------------------------------------
; TA00ISR                                                                 v1.2
;-------------------------------------------------------------------------------
TA00ISR     ;Rutina de servicio de la IRQ del CCR1 (palanca izquierda)
            Intermit    0,TA0CCTL1,TA0CCR1,TA0CCTL3,TA0CCR3,TA0CCTL0,TA0CCR0,TA0CCTL2,
TA0CCR2
```

CRITERIO DE CORRECCIÓN

- Inicialización: 30%

- ISR CCR2/3: 10%
- ISR CCR0/1: 60%